IoT Product Design and Rapid Prototyping

Part C

Brian Rashap

January 2023

Particle Projects from the CLI

Create, Compile, Flash from the CLI

```
particle project create
# give your program a name and select N for default project directory
> A new project has been initialized in directory C:\Users\ddcio\Documents\IoT\HelloCLI

cd HelloCLI/src
emacs HelloCLI.ino   #use your favorite editor (notepad.exe, nano, vim, emacs, etc.)

cat HelloCLI.ino
/*
 * Project HelloCLI
 * Description: First Program Creating without VSCode
 * Author: Brian Rashap
 * Date: 09-JUN-2023
 */

void setup() {
  pinMode(D7,OUTPUT);
}

void loop() {
  digitalWrite(D7,HIGH);
  delay(100);
  digitalWrite(D7,LOW);
  delay(100);
}

particle compile argon --target 4.0.2
particle usb dfu
particle flash --usb .\argon_firmware_1686334897288.bin
```

Particle Libraries

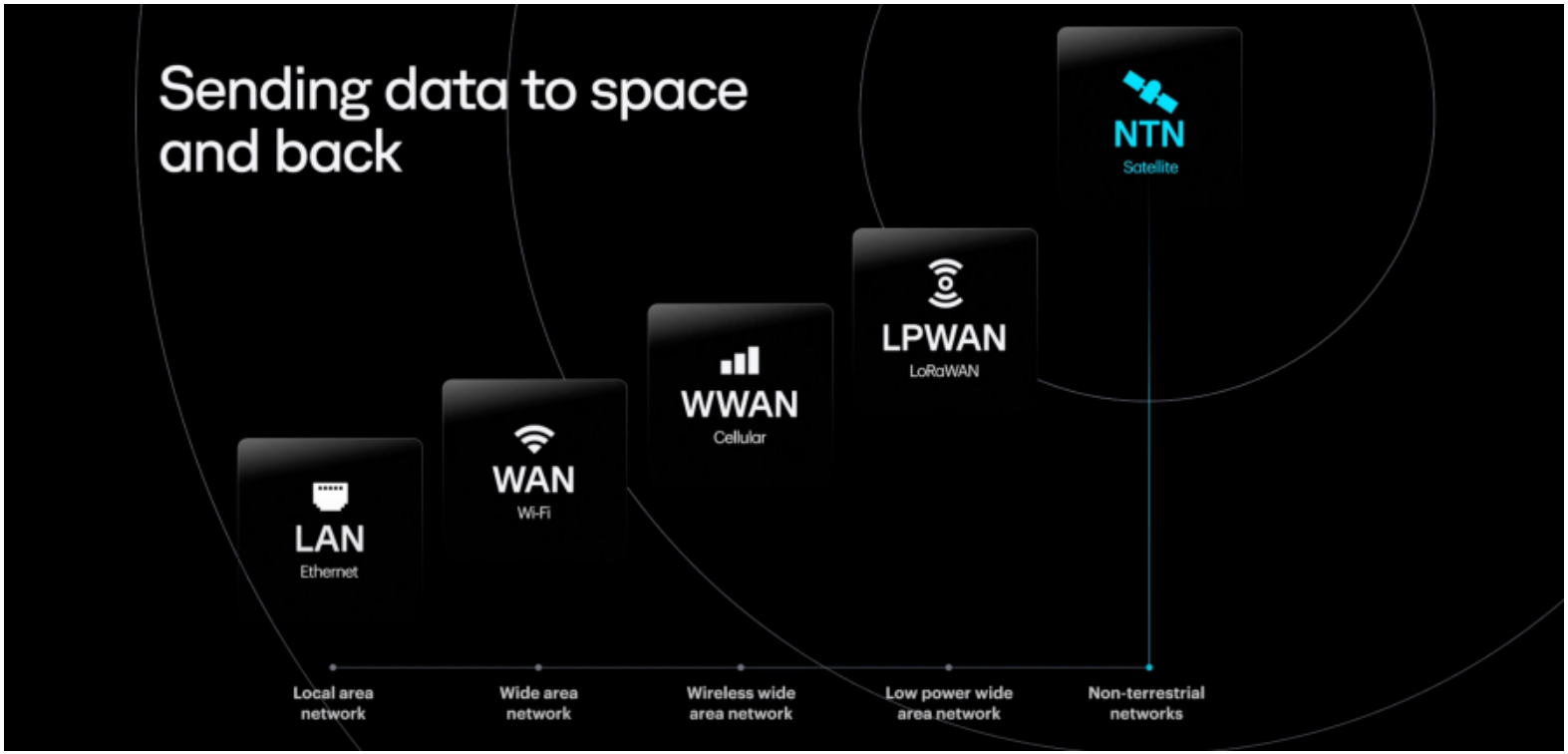
Creating and Publishing Your Own Libraries

```
// Within your project directory, create a library
mkdir mylib
cd mylib
particle library create

// Modify the Project.Properties file, especially version number

// Create your .h, .cpp, and/or examples within the library directory structure

// Upload your library
particle library upload

// Publish your library
particle library publish mylib

// Note: you can upload/publish new versions
//       just change the version number in Project.Properties
```

Non Terrestrial Networks (NTN)

Non Terrestial Networks

Terrestrial networks include cellular networks (2G, 3G, 4G, and 5G), Wi-Fi, LoRA, and low-power wide-area networks (e.g.,LoRaWAN).



Non-terrestrial networks complement traditional terrestrial networks by providing wireless connectivity from airborne or spaceborne platforms, providing the missing link between terrestrial networks.

Non-Embedded C++

HelloWorld in C++

```
// include the standar input-output library
// most include statements need the .h, but iostream is an exception
#include <iostream>

// namespace is used to declare regions with the global space
// std enable the standard console (monitor) and input (keyboard)
using namespace std;

char myName[20];

//main is the first function executed in your cpp code
int main()
{
    cout << "Hello World!!!\n";    // cout = output to console
    cout << "What is your name: ";
    cin >> myName;                 // cin = input from console
    cout << "Hello " << myName << ", I hope you are having a nice day.\n";

    return 0;  // denotes successfully executed
}
```

Instructions for installing and writing C++ code in VSCode:
https://code.visualstudio.com/docs/languages/cpp

The Big Question: What about main()

```
#include <Arduino.h>

extern "C" int main(void)
{
#ifdef USING_MAKEFILE

  // To use Teensy 3.0 without Arduino, simply put your code here.
  // For example:

  pinMode(13, OUTPUT);
  while (1) {
    digitalWriteFast(13, HIGH);
    delay(500);
    digitalWriteFast(13, LOW);
    delay(500);
  }


#else
  // Arduino's main() function just calls setup() and loop()....
  setup();
  while (1) {
    loop();
    yield();
  }
#endif
}
```

Hello World - What a microcontroller sees

HelloWorld.ino

```
void setup() {
  Serial.begin(9600);
  Serial.printf("Hello World! \n");
}

void loop() {}
```

Hex Code and Assembly Language

```
HelloWorld.bin:     file format binary
Disassembly of section .data:
00000000 <.data>:
    101c: bd10        pop {r4, pc}
    101e: 4402        add r2, r0
    1020: 4603        mov r3, r0
    1022: 4293        cmp r3, r2
    1024: d002        beq.n 0x102c
    1026: f803 1b01   strb.w  r1, [r3], #1
    102a: e7fa        b.n 0x1022
    102c: 4770        bx  lr
    102e: 0000        movs  r0, r0
    1030: b538        push  {r3, r4, r5, lr}
```

Useful BASH commands

Redirect from stdout (standard output)

The > and >> signs are used for redirecting the output of a program to something other than stdout (standard output, which is the terminal by default).

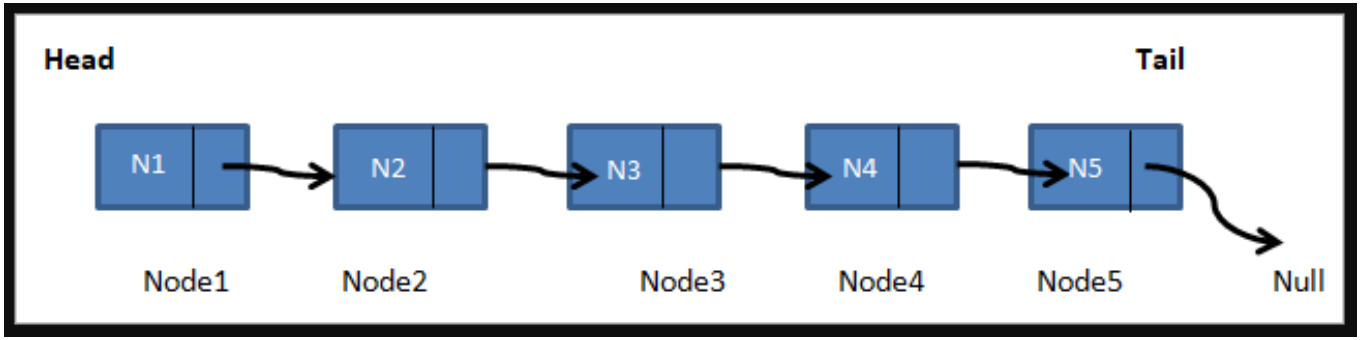
- The >> appends to a file or creates the file if it doesn't exist.
- The > overwrites the file if it exists or creates it if it doesn't exist.

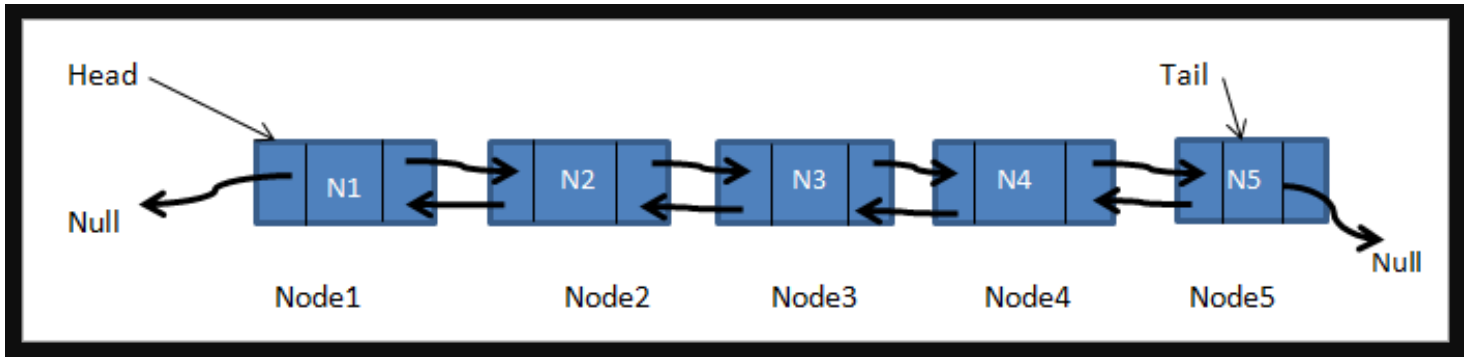
Examples:

```

# Create a file called "allmyfiles.txt" and fill with the directory listing

ls > allmyfiles.txt

# Adds "End the directory listing" to the end of "allmyfiles.txt"

echo "End of directory listing" >> allmyfiles.txt

# Create a zero-byte file with the name "newfile.txt"

> newfile.txt

# Redirect Particle Serial Monitor output to the file "filename.csv"

Particle serial monitor --follow >> filename.csv
```

Linked Lists and Trees

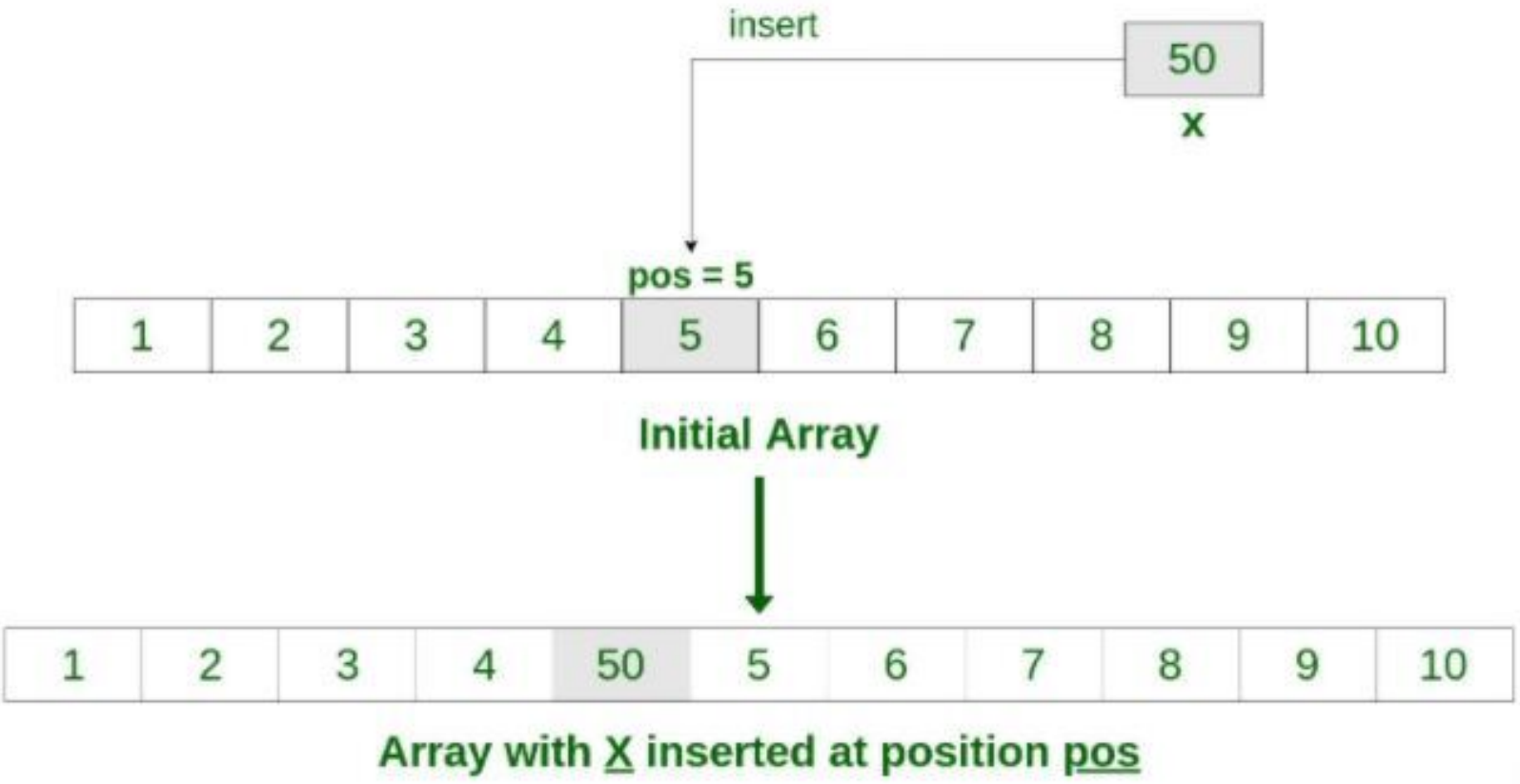
Linked Lists and Doubly Linked Lists



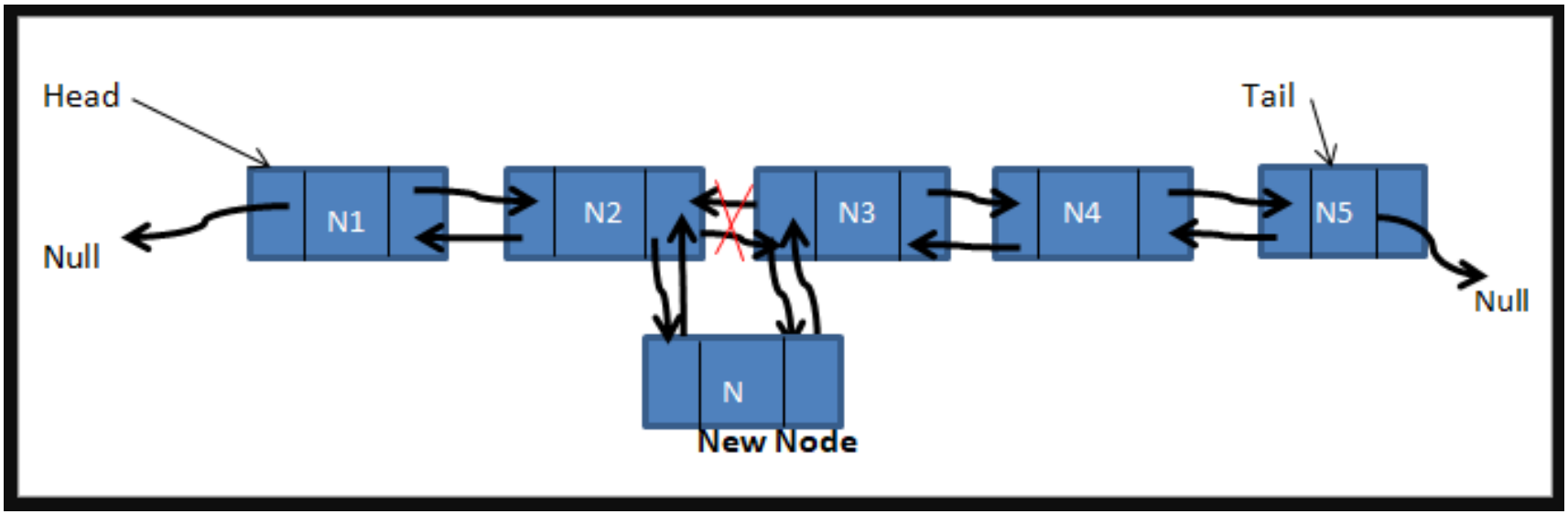



```
struct node {
  struct node *prev;
  int data;
  struct node *next;
};
```

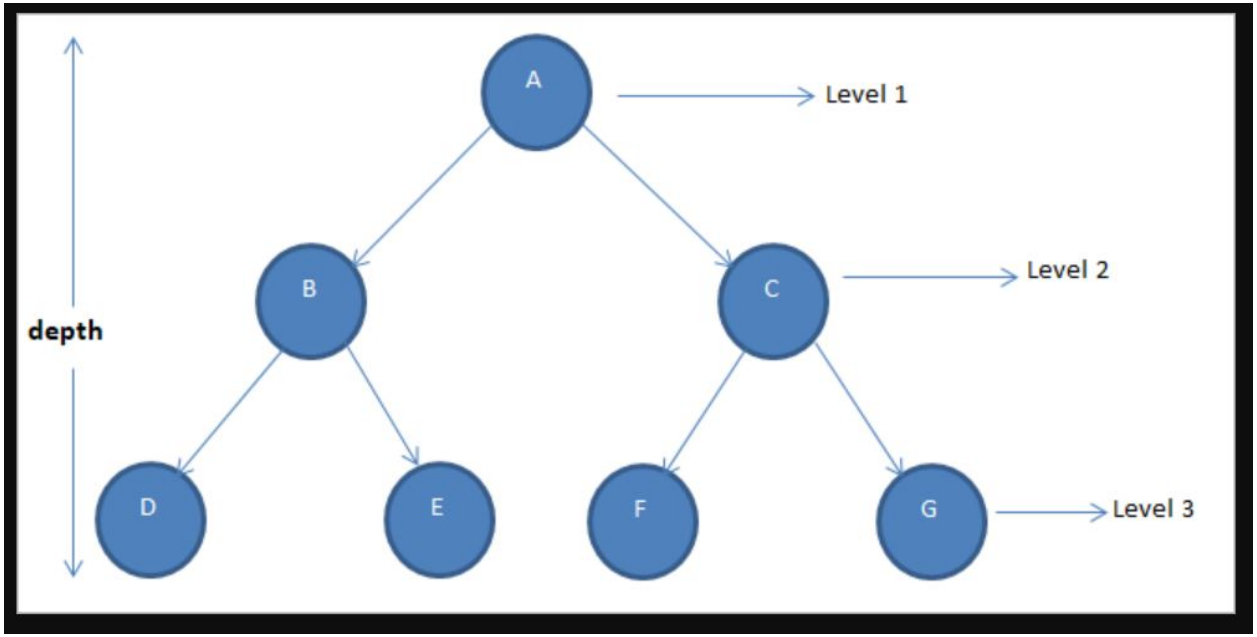
Inserting a "cell" into an Array



Inserting a "cell" into a Linked List

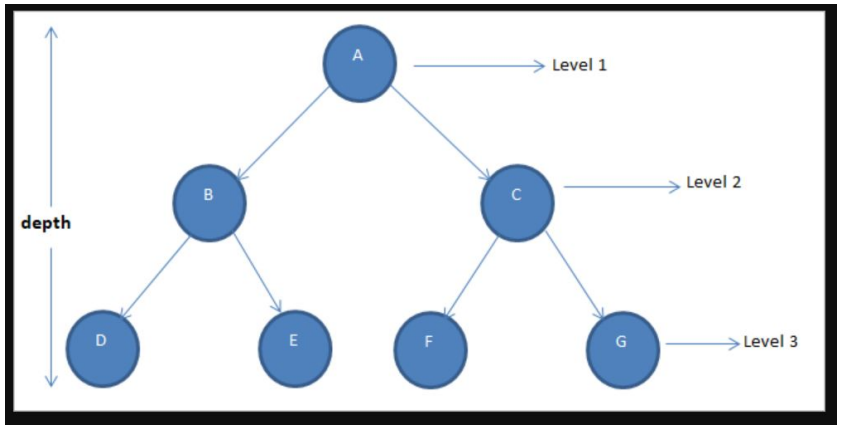


Binary Trees




```
struct bintree_node{
    bintree_node *left;
    bintree_node *right;
    int data;
};
```

Binary Trees



Uses of Binary Trees

- Binary Search
- Hash Trees
- Heaps
- Huffman Coding
- Syntax Tree

```
struct bintree_node{
    bintree_node *left;
    bintree_node *right;
    int data;
};
```

old stuff

Step 1 - OpenWeather

- Create an account at `openweathermap.org`
- Generate an API key at: `https://home.openweathermap.org/api_keys`

You can generate as many API keys as needed for your subscription. We accumulate the total load from all of them.

Key	Name	Status	Actions
07f56344e3fe7a27974a52eb54783b71	Default	Active	
dacc7f9f41f4cca0a274cf925b97a356	IoTClass	Active	

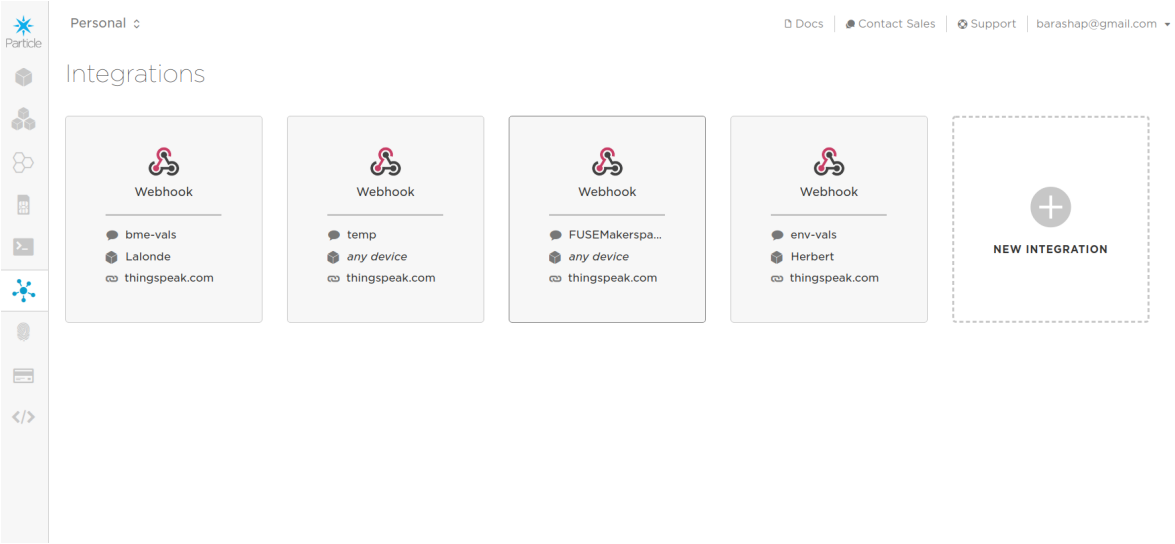
Create key

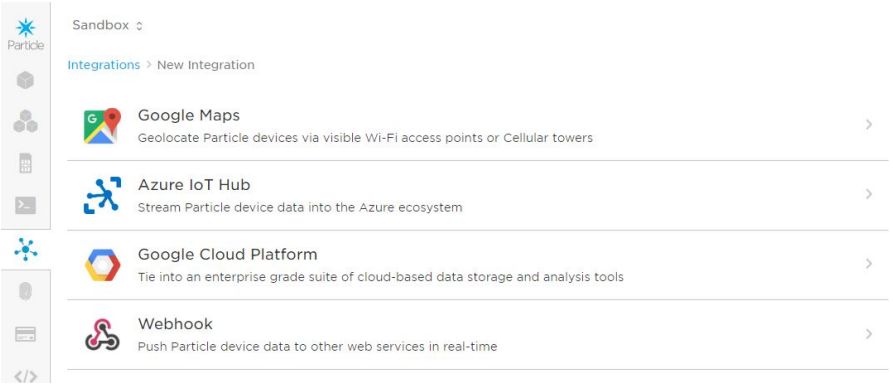
API key name

Generate

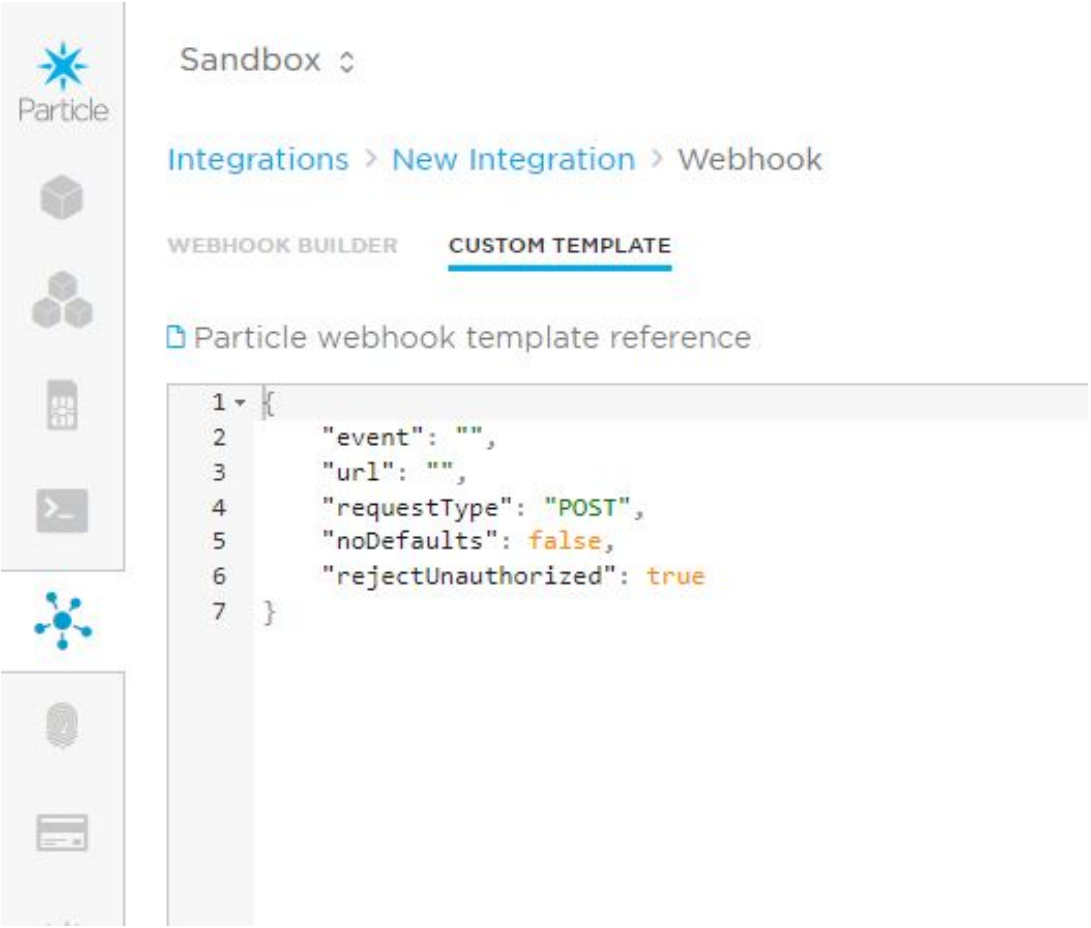
Step 2 - Create Webhook

From `console.particle.io`:





Step 3 - Select Custom Template



Step 4 - Update Custom Template with API format

Adapted from `https://openweathermap.org/api/one-call-api` and `https://openweathermap.org/current`

```
{
    "name": "GetWeatherData for openweathermap.org",
    "event": "GetWeatherData",
    "responseTopic": "{{PARTICLE_DEVICE_ID}}/{{PARTICLE_EVENT_NAME}}",
    "url": "https://api.openweathermap.org/data/2.5/weather",
    "requestType": "GET",
    "noDefaults": true,
    "rejectUnauthorized": true,
    "responseTemplate": "{\"lon\":{{coord.lon}},\"lat\":{{coord.lat}},\"temp\":{{main.
     temp}},\"wind\":{{wind.speed}},\"conditions\":\"{{weather.0.main}}\"}",
    "unchunked": false,
    "data_url_response_event": false,
    "query": {
        "lat": "{{lat42}}",
        "lon": "{{lon42}}",
        "exclude": "minutely,hourly,alerts",
        "units": "metric",
        "appid": "YOUR_APP_ID"  // replace YOUR_APP_ID with your ID
    }
}
```

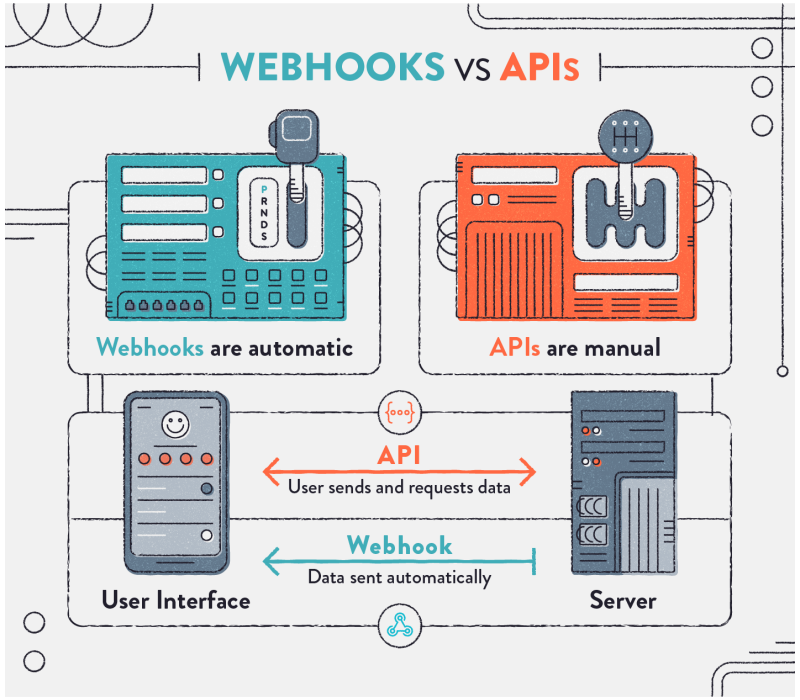
Step 5 - Particle Code

```
#include "Particle.h"
// Weather Constants and Variables
const char *EVENT_NAME = "GetWeatherData";
float lat42,lon42;
float tempC,wind;
int lastWeather;
String condition;
// Time Variables
int hours, minutes, lasthour, lastminute;

void subscriptionHandler(const char *event, const char *data);

void setup() {
    Serial.begin(9600);
    waitFor(Serial.isConnected,5000);
    delay(500);
    String subscriptionName = String::format("%s/%s/", System.deviceID().c_str(),
     EVENT_NAME);
    Particle.subscribe(subscriptionName, subscriptionHandler, MY_DEVICES);
    Serial.printf("subscribing to %s\n", subscriptionName.c_str());

    Time.beginDST();
    Time.zone(-7);
    Particle.syncTime();
    lastWeather = -9999999;
    lastminute = Time.minute();

    lat42 = 35.08392;
    lon42 = -106.64787;
}
```

Step 5 - Particle Code

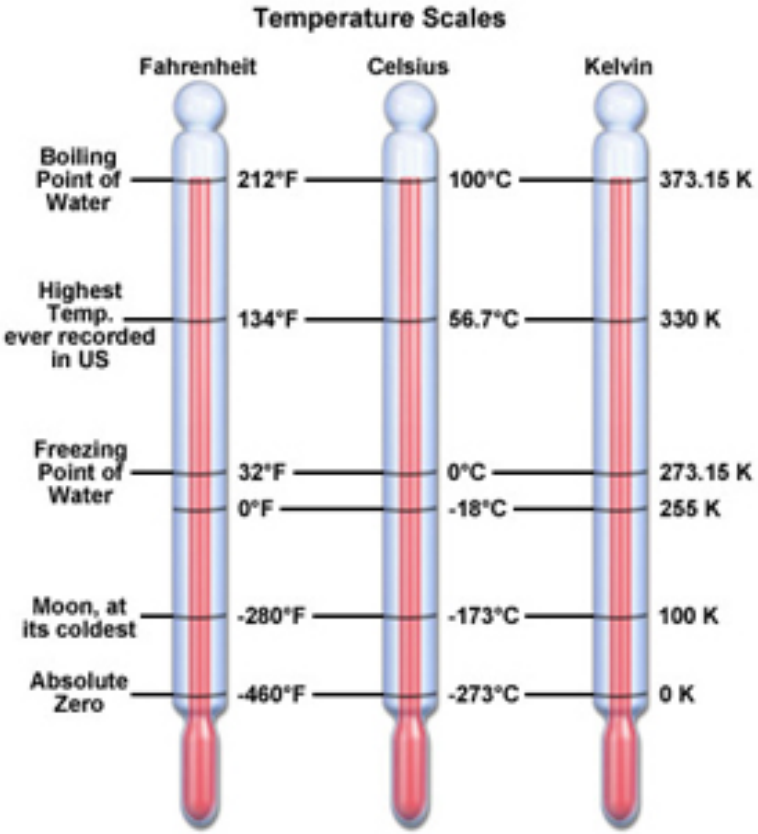
```
void loop() {
    minutes = Time.minute();
    hours = Time.hour();
    if((minutes != lastminute)&&(minutes%10 == 0)) {
        Particle.publish(EVENT_NAME, String::format("{\"lat42\":%0.5f,\"lon42\":%0.5f}",
 lat42, lon42), PRIVATE);
        lastminute=Time.minute();
    }
}
void subscriptionHandler(const char *event, const char *data) {
    JSONValue outerObj = JSONValue::parseCopy(data);
    JSONObjectIterator iter(outerObj);
    Serial.printf("\n\n\nWeather at %2i:%02i\n",Time.hour(),Time.minute());
    while(iter.next()) {
        if (iter.name() == "lat") {
            Serial.printf("Latitude: %0.6f\n", iter.value().toDouble());
        }
        if (iter.name() == "lon") {
            Serial.printf("Longitude: %0.6f\n", iter.value().toDouble());
        }
        if (iter.name() == "temp") {
            tempC = iter.value().toDouble();
            Serial.printf("Temperature: %0.2f(C)\n", iter.value().toDouble());
        }
        if (iter.name() == "wind") {
            Serial.printf("Wind Speed: %0.2f (m/s)\n", iter.value().toDouble());
        }
        if (iter.name() == "conditions") {
            Serial.printf("Conditions: %s\n", (const char *)iter.value().toString());
        }
    }}
```

Assignment: L09_03_GetWeather



1. Using the OpenWestherMap webhook get the outside weather conditions.
2. Display the OLED:
   - GPS location (hard coded)
   - Indoor conditions from BME280
   - Current outdoor conditions.

Mapping (or Converting)



Mapping is the conversion from one set of units to another. For example converting from Celsius to Fahrenheit:

$$Temp(^\circ F) = \frac{9}{5} * Temp(^\circ C) + 32$$

C++ provides us with a function to do this mapping:
newVal = map(value, fromLow, fromHigh, toLow, toHigh);

For example:
tempF = map(tempC,0,100,32,212);